

Ministry of Education of Moldova
Technical University of Moldova
Faculty "Computers, Informatics and Microelectronics"

Report

Laboratory work nr. 4
on Embedded systems

Performed by:
st. gr. FAF-141

D. Lupei

Verified by:
assoc. prof.,

A. Bragarenco

Chisinau -2016

Topic

Pulse width modulation. Controlling motor with H-Bridge

Objectives

- Implement keyboard control with USART and Virtual Terminal
- Implement Pulse-width modulation
- Implement H-Bridge circuit

Tasks

Write a C program and schematics for a car using Universal asynchronous receiver/transmitter, h-bridge, pulse width modulation. Use keyboard as control for wheels. Car should be able to steer, increase velocity, decrease velocity, stop or free wheeling

Short theory

DC electric motors are arguably the most commonly used method for locomotion in mobile robots. DC motors are clean, quiet, and can produce sufficient power for a variety of tasks. They are much easier to control than pneumatic actuators, which are mainly used if very high torques are required and umbilical cords for external pressure pumps are available – so usually not an option for mobile robots.

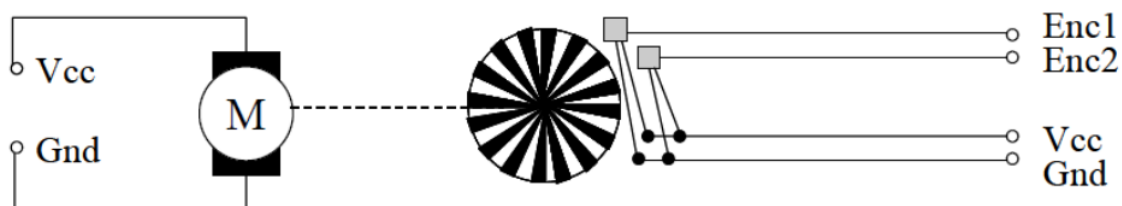


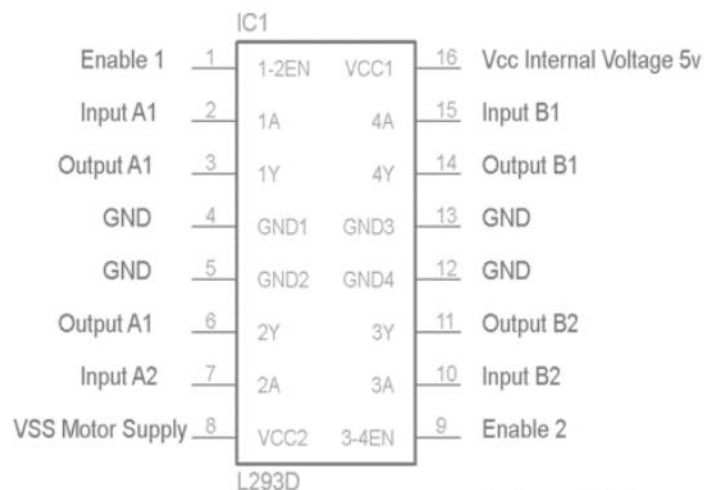
Fig.1 DC Motor

For most applications we want to be able to do two things with a motor:

1. run it in forward (clockwise) and backward (counterclockwise) directions
2. modify its speed
- 3.

Motor driver (L293D)

First, to allow the motor to run clockwise and counterclockwise it is needed a motor drive. In this project was used the L293D motor drive which implements the H-Bridge circuit. L293D is a typical Motor driver or Motor Driver integrated circuit which is used to drive direct current on either direction. It is a 16-pin IC which can control a set of two DC motors simultaneously in any direction.

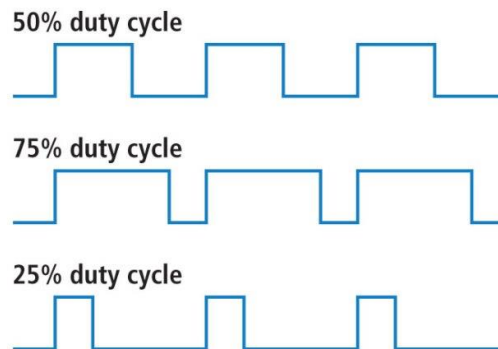


Pulse width modulation (PWM) is a fancy term for describing a type of digital signal. Pulse width modulation is used in a variety of applications including sophisticated control circuitry. A common way to use them is to control dimming of RGB LEDs or to control the direction of a servo motor. We can accomplish a range of results in both applications because pulse width modulation allows us to vary how much time the signal is high in an analog fashion. While the signal can only be high (usually 5V) or low (ground) at any time, we can change the proportion of time the signal is high compared to when it is low over a consistent time interval.

Duty Cycle

When the signal is high, we call this “on time”. To describe the amount of “on time”, we use the concept of duty cycle. Duty cycle is measured in percentage. The percentage duty cycle specifically describes the percentage of time a digital signal is on over an interval or period of time. This period is the inverse of the frequency of the waveform.

If a digital signal spends half of the time on and the other half off, we would say the digital signal has a duty cycle of 50% and resembles an ideal square wave. If the percentage is higher than 50%, the digital signal spends more time in the high state than the low state and vice versa if the duty cycle is less than 50%. Here is a graph that illustrates these three scenarios:



50%, 75%, and 25% Duty Cycle Examples

100% duty cycle would be the same as setting the voltage to 5 Volts (high). 0% duty cycle would be the same as grounding the signal.

H-Bridge

An *H bridge* is an electronic circuit that enables a voltage to be applied across a load in either direction. These circuits are often used in robotics and other applications to allow DC

motors to run forwards or backwards. Most DC-to-AC converters (power inverters), most AC/AC converters, the DC-to-DC push-pull converter, most motor controllers, and many other kinds of power electronics use H bridges. In particular, a bipolar stepper motor is almost invariably driven by a motor controller containing two H bridges.

The H-bridge arrangement is generally used to reverse the polarity/direction of the motor, but can also be used to 'brake' the motor, where the motor comes to a sudden stop, as the motor's terminals are shorted, or to let the motor 'free run' to a stop, as the motor is effectively disconnected from the circuit. The following table summarises operation, with S1-S4 corresponding to the diagram above.

H bridges are available as integrated circuits, or can be built from discrete components.

The term *H bridge* is derived from the typical graphical representation of such a circuit. An H bridge is built with four switches (solid-state or mechanical). When the switches S1 and S4 (according to the first figure) are closed (and S2 and S3 are open) a positive voltage will be applied across the motor. By opening S1 and S4 switches and closing S2 and S3 switches, this voltage is reversed, allowing reverse operation of the motor.

Using the nomenclature above, the switches S1 and S2 should never be closed at the same time, as this would cause a short circuit on the input voltage source. The same applies to the switches S3 and S4. This condition is known as shoot-through.

| S1 | S2 | S3 | S4 | Result |
|----|----|----|----|-------------------|
| 1 | 0 | 0 | 1 | Motor moves right |
| 0 | 1 | 1 | 0 | Motor moves left |
| 0 | 0 | 0 | 0 | Motor coasts |
| 0 | 1 | 0 | 1 | Motor brakes |
| 1 | 0 | 1 | 0 | Motor brakes |
| 1 | 1 | 0 | 0 | Short circuit |
| 0 | 0 | 1 | 1 | Short circuit |
| 1 | 1 | 1 | 1 | Short circuit |

Work Process

1. Environment setup by creating the following files: main.c, car_2wd.c, motor.c, pwm.c, gpio.c, hbridge.c, uart_stdio.c, car_2wd.h, motor.h, pwm.h, hbridge.h, uart_stdio.h,
2. UART driver initializes serial IO for UART and it used as keyboard controller.

```
void uart_stdio_Init(void);
int uart_PutChar(char c, FILE *stream);
char uart_ReadChar();
```

3. Car_2w driver has a descriptor which is composed from two DC motor and methods to control de car from the keyboard

```
Car* CAR_create(Motor *leftMotor, Motor *rightMotor);
void CAR_start(Car *descriptor);
void CAR_stop(Car *descriptor);
void CAR_left(Car *descriptor);
void CAR_right(Car *descriptor);
void CAR_forward(Car *descriptor);
void CAR_backward(Car *descriptor);
void CAR_brake(Car *descriptor);
void CAR_calibrate_speed(Car *descriptor, uint8_t increment);
```

4. Hbridge driver

```
HBridge* HBRIDGE_create(GPIO *en, GPIO *in1, GPIO *in2);
void HBRIDGE_init(HBridge *descriptor);
void HBRIDGE_enable(HBridge *descriptor);
void HBRIDGE_disable(HBridge *descriptor);
void HBRIDGE_set_operation(HBridge *descriptor, HBridge_Operation operation);
```

5. Motor Driver

```
Motor* MOTOR_create(HBridge *descriptor, void (*pwm)(uint8_t));
void MOTOR_start(Motor *descriptor);
void MOTOR_stop(Motor *descriptor);
void MOTOR_set_direction(Motor *descriptor, Motor_Direction direction);
void MOTOR_set_speed(Motor *descriptor, uint8_t speed);
void MOTOR_reset_speed(Motor *descriptor);
void MOTOR_brake(Motor *descriptor);
```

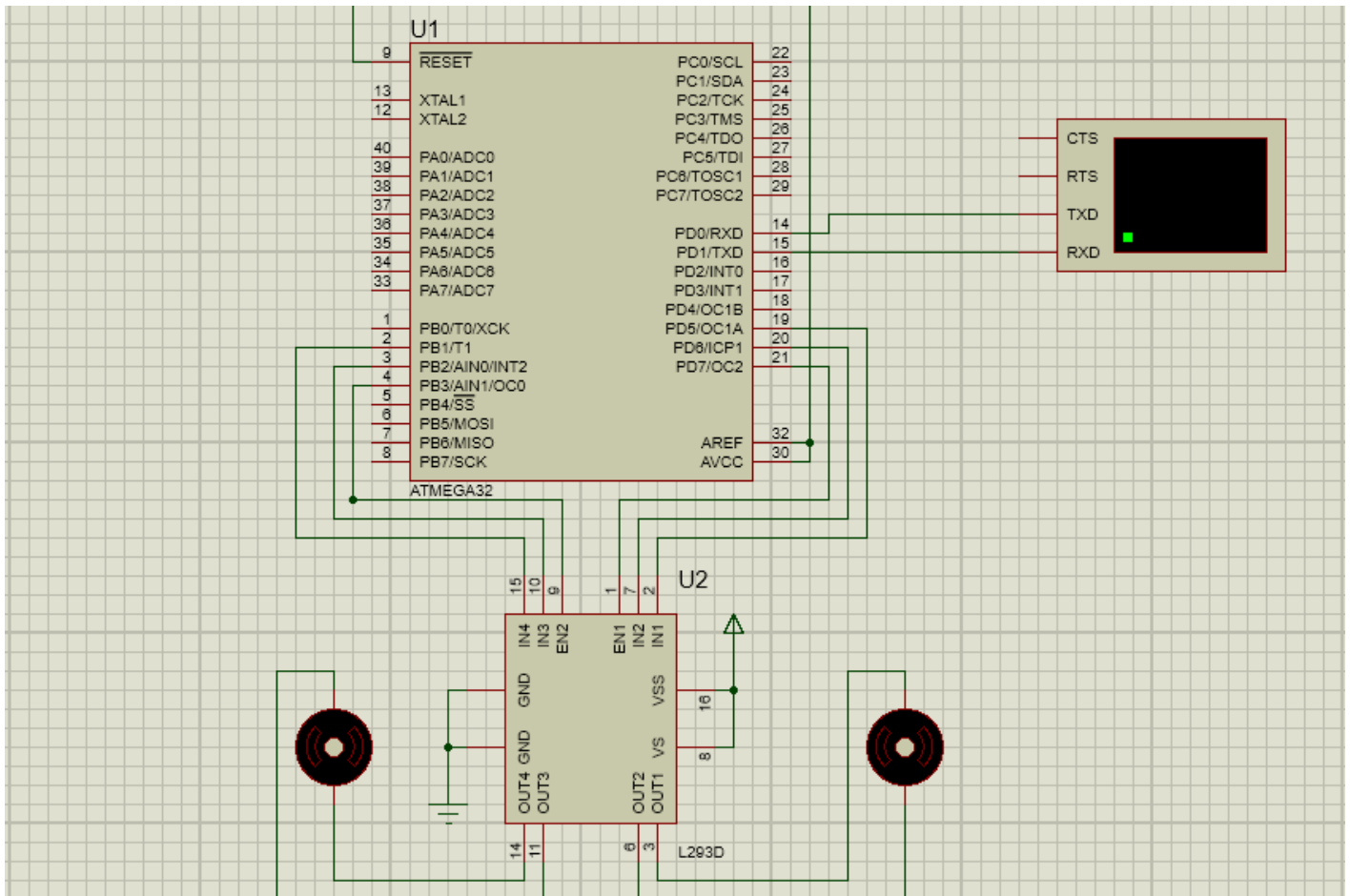
6. PWM driver

```
Sets first 8bit timer for PWM with phase
void pwm_0_set(uint8_t time_on);
Sets second 8bit timer for PWM with phase
void pwm_2_set(uint8_t time_on);
```

7. Building the solution and importing the .elf resulting file in Proteus

Circuit Scheme

I've connected a H-Bridge circuit L293 to micro-controller and connected 2 DC motors to H-Bridge



Conclusion:

In this laboratory work has been learnt the principles of how a DC motor interfaces with ATmega32. The interesting part was controlling the motor's rotation direction and its speed. The first thing was accomplished using H-Bridge circuit from the motor driver L293D, and the second thing – using PWM

Bibliography

- M.A. Mazidi, “*AVR microcontroller and embedded system using C and assembly language*”, 2009
- Thomas Bräunl, “*Embedded Robotics*”
- www.effusiontech.com/pwm-in-avr.html
- www.rakeshmondal.info/L293D-Motor-Driver

Appendix

main.c

```
#include <avr/io.h>
#include <avr/delay.h>
#include <drivers/hbridge.h>
#include "drivers/uart_stdio.h"
#include "drivers/pwm.h"
#include "gpio.h"
#include "drivers/motor.h"
#include "drivers/car_2wd.h"

Car *car;

void create(){

    HBridge *leftHBridge = HBRIDGE_create(
        GPIO_create(&DDRB,&PORTB,&PINB,3),
        GPIO_create(&DDRB,&PORTB,&PINB,2),
        GPIO_create(&DDRB,&PORTB,&PINB,1)
    );
    Motor *leftMotor = MOTOR_create(leftHBridge,&pwm_0_set);

    HBridge *rightHBridge = HBRIDGE_create(
        GPIO_create(&DDRD,&PORTD,&PIND,7),
        GPIO_create(&DDRD,&PORTD,&PIND,5),
        GPIO_create(&DDRD,&PORTD,&PIND,6)
    );
    Motor *rightMotor = MOTOR_create(rightHBridge,&pwm_2_set);

    car = CAR_create(leftMotor,rightMotor);
}

int main() {
    uart_stdio_Init();

    char key;
    create();

    while(1){
        printf("\nEnter command:");
        key = getchar();

        switch(key){
            case 'a' :
                CAR_left(car);
                break;
            case 'w':
                CAR_forward(car);
                break;
            case 'd':
                CAR_right(car);
                break;
            case 's':
                CAR_backward(car);
                break;
        }
    }
}
```

```

        case 'p':
            CAR_start(car);
            break;
        case 'l':
            CAR_stop(car);
            break;
    }
}

return 0;
}

```

gpio.c

```

#include "gpio.h"
#include <stdlib.h>
#include "utils.h"

GPIO* GPIO_create(uint8_t volatile *ddr, uint8_t volatile *port, uint8_t volatile
*pin, uint8_t id){
    GPIO *descriptor = malloc(sizeof(GPIO));
    descriptor->ddr = ddr;
    descriptor->port = port;
    descriptor->pin = pin;
    descriptor->id = id;

    return descriptor;
}

void GPIO_set_mode(GPIO *descriptor, GPIO_Mode mode){
    switch(mode){
        case GPIO_MODE_INPUT:
            bit_set_0(descriptor->ddr, descriptor->id);
            bit_set_1(descriptor->port, descriptor->id);
            break;

        case GPIO_MODE_OUTPUT:
            bit_set_1(descriptor->ddr, descriptor->id);
            break;
    }
}

void GPIO_write(GPIO *descriptor, GPIO_Value value){
    if(value == GPIO_LOW){
        bit_set_0(descriptor->port, descriptor->id);
    } else {
        bit_set_1(descriptor->port, descriptor->id);
    }
}

GPIO_Value GPIO_read(GPIO *descriptor){
    uint8_t value = (*descriptor->pin) & (1 << descriptor->id);

    return value == 0 ? GPIO_LOW : GPIO_HIGH;
}

```

Hbridge.c

```
#include <drivers/hbridge.h>
#include "utils.h"
#include <stdlib.h>

HBridge* HBRIDGE_create(GPIO *en,GPIO *in1,GPIO *in2){
    HBridge *descriptor = malloc(sizeof(HBridge));
    descriptor->en = en;
    descriptor->in1 = in1;
    descriptor->in2 = in2;

    return descriptor;
}

void HBRIDGE_init(HBridge *descriptor){
    GPIO_set_mode(descriptor->en,GPIO_MODE_OUTPUT);
    GPIO_set_mode(descriptor->in1,GPIO_MODE_OUTPUT);
    GPIO_set_mode(descriptor->in2,GPIO_MODE_OUTPUT);
}

void HBRIDGE_enable(HBridge *descriptor){
    GPIO_write(descriptor->en,GPIO_HIGH);
}

void HBRIDGE_disable(HBridge *descriptor){
    GPIO_write(descriptor->en,GPIO_LOW);
}

void HBRIDGE_set_operation(HBridge *descriptor,HBridge_Operation operation){
    uint8_t in1 = 0;
    uint8_t in2 = 0;
    switch(operation){
        case HBRIDGE_OPERATION_LEFT:
            in1 = 1;
            break;
        case HBRIDGE_OPERATION_RIGHT:
            in2 = 1;
            break;
        case HBRIDGE_OPERATION_BREAK:
            in1 = 1;
            in2 = 1;
            break;
    }

    GPIO_write(descriptor->in1,in1 ? GPIO_HIGH : GPIO_LOW);
    GPIO_write(descriptor->in2,in2 ? GPIO_HIGH : GPIO_LOW);
}
```

pwm.c

```
#include <avr/io.h>
#include "drivers/pwm.h"
#include "utils.h"

void pwm_0_set(uint8_t time_on){
    // set timer0 for fast pwm
    bit_set_1(&TCCR0,WGM00);
    bit_set_1(&TCCR0,WGM01);

    // set clear on compare match and set on top
    bit_set_1(&TCCR0,COM01);

    // set prescaler 256
    bit_set_1(&TCCR0,CS02);

    bit_set_1(&DDRB,PB3);

    // set compare value
    OCR0 = time_on;

    // reset counter
    TCNT0 = 0;
}

void pwm_2_set(uint8_t time_on){

    // set timer2 for fast pwm
    bit_set_1(&TCCR2,WGM20);
    bit_set_1(&TCCR2,WGM21);

    // set clear on compare match and set on top
    bit_set_1(&TCCR2,COM21);

    // set prescaler 256
    bit_set_1(&TCCR2,CS21);
    bit_set_1(&TCCR2,CS22);

    bit_set_1(&DDRD,PD7);

    // set compare value
    OCR2 = time_on;

    // reset counter
    TCNT2 = 0;
}
```

uart_stdio.c

```
#include "drivers/uart_stdio.h"

FILE uart_output = FDEV_SETUP_STREAM(uart_PutChar, NULL, _FDEV_SETUP_WRITE);
FILE uart_input = FDEV_SETUP_STREAM(NULL, uart_ReadChar, _FDEV_SETUP_READ);

void uart_stdio_Init(void)
{
    stdout = &uart_output;
    stdin = &uart_input;

    #if F_CPU < 2000000UL && defined(U2X)
        UCSRA = _BV(U2X); /* improve baud rate error by using 2x clk */
        UBRR1 = (F_CPU / (8UL * UART_BAUD)) - 1;
    #else
        UBRR1 = (F_CPU / (16UL * UART_BAUD)) - 1;
    #endif
    UCSRB = _BV(TXEN) | _BV(RXEN); /* tx/rx enable */
}

int uart_PutChar(char c, FILE *stream)
{
    if (c == '\n')
        uart_PutChar('\r', stream);

    while (~UCSRA & (1 << UDRE));
    UDR = c;

    return 0;
}

char uart_ReadChar()
{
    //Wait untill a data is available
    while(!(UCSRA & (1<<RXC)))
    {
        //Do nothing
    }

    //Now USART has got data from host
    //and is available is buffer

    return UDR;
}
```

motor.c

```
#include "motor.h"
#include <stdlib.h>
#define MAX_PWM_VALUE 255

Motor* MOTOR_create(HBridge *hbridge,void (*pwm)(uint8_t)){
    Motor *descriptor = malloc(sizeof(Motor));
    descriptor->hbridge = hbridge;
    descriptor->speed = 0;
    descriptor->pwm = pwm;

    HBRIDGE_init(hbridge);

    return descriptor;
}

void MOTOR_start(Motor *descriptor){
    MOTOR_reset_speed(descriptor);
}

void MOTOR_stop(Motor *descriptor){
    descriptor->pwm(0);
}

void MOTOR_set_direction(Motor *descriptor,Motor_Direction direction){
    switch(direction){
        case MOTOR_DIRECTION_LEFT:
            HBRIDGE_set_operation(descriptor->hbridge,HBRIDGE_OPERATION_LEFT);
            break;
        case MOTOR_DIRECTION_RIGHT:
            HBRIDGE_set_operation(descriptor->hbridge,HBRIDGE_OPERATION_RIGHT);
            break;
    }
    MOTOR_reset_speed(descriptor);
}

void MOTOR_brake(Motor *descriptor){
    HBRIDGE_set_operation(descriptor->hbridge,HBRIDGE_OPERATION_BREAK);
    descriptor->pwm(MAX_PWM_VALUE);
}

void MOTOR_set_speed(Motor *descriptor,uint8_t speed){
    if(speed < 0)
        speed = 0;
    if(speed > 100)
        speed = 100;
    descriptor->speed = speed;
    MOTOR_reset_speed(descriptor);
}

void MOTOR_reset_speed(Motor *descriptor){
    uint8_t convertedSpeed = descriptor->speed*MAX_PWM_VALUE/100;
    descriptor->pwm(convertedSpeed);
}
```

car_2wd.c

```
#include "drivers/car_2wd.h"
#include <stdlib.h>

Car* CAR_create(Motor *leftMotor, Motor *rightMotor){
    Car *descriptor = malloc(sizeof(Car));
    descriptor->leftMotor = leftMotor;
    descriptor->rightMotor = rightMotor;

    return descriptor;
}

void CAR_start(Car *descriptor){
    MOTOR_start(descriptor->leftMotor);
    MOTOR_start(descriptor->rightMotor);
}

void CAR_stop(Car *descriptor){
    MOTOR_stop(descriptor->leftMotor);
    MOTOR_stop(descriptor->rightMotor);
}

void CAR_left(Car *descriptor){
    MOTOR_set_speed(descriptor->leftMotor, descriptor->leftMotor->speed-1);
    MOTOR_set_speed(descriptor->rightMotor, descriptor->rightMotor->speed+1);
}

void CAR_right(Car *descriptor){
    MOTOR_set_speed(descriptor->rightMotor, descriptor->rightMotor->speed-1);
    MOTOR_set_speed(descriptor->leftMotor, descriptor->leftMotor->speed+1);
}

void CAR_forward(Car *descriptor){
    CAR_calibrate_speed(descriptor, 1);
    MOTOR_set_direction(descriptor->leftMotor, MOTOR_DIRECTION_RIGHT);
    MOTOR_set_direction(descriptor->rightMotor, MOTOR_DIRECTION_RIGHT);
}

void CAR_backward(Car *descriptor){
    CAR_calibrate_speed(descriptor, 1);
    MOTOR_set_direction(descriptor->leftMotor, MOTOR_DIRECTION_LEFT);
    MOTOR_set_direction(descriptor->rightMotor, MOTOR_DIRECTION_LEFT);
}

void CAR_brake(Car *descriptor){
    MOTOR_brake(descriptor->leftMotor);
    MOTOR_brake(descriptor->rightMotor);
}

void CAR_calibrate_speed(Car *descriptor, uint8_t increment){
    int new_speed = (descriptor->leftMotor->speed + descriptor->rightMotor->speed) / 2
+ increment;
    MOTOR_set_speed(descriptor->leftMotor, new_speed);
    MOTOR_set_speed(descriptor->rightMotor, new_speed);
}
```